SOFTWARE TESTING OF ATTITUDE AND ORBIT CONTROL COMPUTER (AOCC) OF INSAT-2E

S. Udupa, FitzgeraldJ.Archibald, M.M. Nayak, S.Lakshmi,
V.K. Agrawal, N.K. Malik

Control Systems Group
ISRO Satellite Centre
Bangalore – 560 017

91

Abstract

Spacecraft Attitude and Orbit Control System is a mission critical real time embedded system. The special characteristics of real time systems (time – dependent, asynchronous) offer major challenges when testing is to be conducted. In many situations, test data provided, when a real time system is in one state will result in proper processing, while the same data provided when the system is in a different state may lead to error. To validate certain type of computation bound equations, we may have to perform exhaustive testing for even more than 90,000 iterations. This paper brings out the approach and experiences in testing the onboard software.

1. INTRODUCTION

The INSAT 2E is an operational system designed and developed for telecommunication, television and radio broadcasting and meteorological data collection and utilization. The various systems in the spacecraft are structures, propulsion, communication payload, Attitude & Orbit Control System(AOCS), Telemetry & Tele-command (TTC), Power System, deployment mechanism and thermal system

AOCS uses the 3-axis body stabilized momentum biased system with momentum/reaction wheel to provide a stable platform for communication and meteorological applications. The AOCS provides the capability of the 3-axis attitude control. The AOCS consists of MIL-STD-1750A based Attitude & Orbit Control Computer(AOCC) which receives attitude data from sensors, computes the necessary control torque commands and outputs to the actuators. The different modes of operation are launcher phase, transfer orbit operations (South Sun Acquisition, Transfer Orbit Earth Acquisition, Liquid Apogee Motor mode), synchronous orbit operations (Synchronous Orbit East West Sun Acquisition, Synchronous Orbit Earth Acquisition, On-Orbit, Station Keeping) & safe mode. In addition to the control functions, as mentioned above, AOCC also performs related command & monitoring functions. The notable feature of an embedded

system is that there is a large degree of coupling between hardware and software, so is the case of the spacecraft onboard computer. Sound methodologies are adopted for developing software for this embedded system. Details of software development methodology is given in ref [1][4]. Here, we are giving only an outline of the development. In the requirement phase, software requirements are generated based on the spacecraft requirements and control requirements. In design phase, the structure of software is designed, which includes module identification, module interface design and associated reviews. In development phase, code generation and review of codes is carried out based on certain guidelines. In testing phase, all efforts are at detecting software errors by devising different methods and combination of test vectors. This paper gives briefly description of testing philosophy, various methods available and emphasis on the testing methodology followed by illustrative examples.

2. BACKGROUND

Reliability of a software depends upon the methodology adopted for software development. In any of the software methodology, testing plays an important role. A good amount of testing improves the quality of Software by uncovering the software bugs and proves that software meets design specification and hence requirements. Particularly, the testing plays a very important measure for the application like ours, where it is not possible to repair the software and it is mission critical. Any bug in the software may result in mission disaster. In fact, for a critical system, testing takes about 50% of the development time. As the software becomes complex, the test procedure also becomes complex. According to Glen Myers [5], the following rules serve as testing objective.

(i) Testing is a process of executing a program with the intent of finding an error.

(ii) A good test case is one that has a high probability of finding an as yet undiscovered error.

(iii) A successful test is one that uncovers an as yet undiscovered error.

The representation of a testing process can be found in ref[5].

Based on the software configuration and test configuration test cases are prepared. The system is then tested with these test cases. The results of these test cases are logged and are evaluated based on the expected test results. The errors are then corrected. The observed errors can be used as a reliability prediction of the software, depending upon the reliability model.

A systematic testing involves a systematic test procedure at various levels of software developed. The testing has to be carried out at various levels of development. A typical software test stage configuration can be found in ref [9]. In this software classification, the software is divided into several stages. A software system is divided into software subsystem, each subsystem is divided into software modules. Each of these software modules are in turn divided into procedure or function. The testing starts from the procedure level and ends at the system level . This testing process is iterative. The errors found at each stage is fed back to the previous stage for correction and retesting. A proper classification of each of the software component helps in independent testing at each stage.

For an efficient testing of a complex system, proper test plan strategy is needed. In general, the testing process commences in parallel with the software development process. The test plan preparation starts with the system requirement phase itself. A typical test plan strategy linked with the software development cycle can be found in ref [9].

In a more complex and very large system, different test strategy may be used for different parts of the system. From the strategy view point, test vectors are designed in three broad types: Top-down approach, bottom-up approach and thread testing [9]. These are briefly described here.

In **top-down testing** method, testing begins at subsystem level with modules represented by stubs. After all the subsystem are tested, modules are tested in the same way by representing the functions and procedures with stubs. Finally, program components are replaced by the actual code and tested. In this

method, the modules get tested as and when they are developed, without waiting for the lower level components to be developed. This method is very difficult to adopt for complex systems, as the stub development consumes more time.

In **bottom-up testing** method, lower level modules are tested first and then higher level modules are tested in upward manner. Test drivers are to be written to simulate the component environment.

In **thread testing** method, each process is tested individually by top-down/ bottom-up methods. This is largely adopted for systems, where the processes are event based and have time dependent interaction between processes.

The success of testing largely depends upon the suitability of test cases. Our objective in designing test cases is to uncover different errors with a minimal time and efforts. Different criteria are used for test case generation at different levels of testing. System level testing and acceptance level testing are concerned with functional and system specification validation/verification, whereas procedure, module and subsystems level are concerned with detection of hidden defects. It is often very time consuming to test all possible combination of paths. As there may not be a definite rule, certain heuristics are normally used in selection of test cases. For example, we should check all types of computation, maximum and minimum value of input variable, overflow and underflow expected conditions, counter loops etc. The test case generation is also linked with the kind of testing.

Testing is classified as

(i) white box testing and

(ii) black box testing [5]

White box testing is applied to the early stages of software development. The method uses the control structure of the procedure design to derive the test cases. Here the main emphasis on the basis path testing and control structure testing.

Test cases of black box testing are designed with the knowledge of product specification. The tests are conducted to demonstrate the functions of

the product, that they meet the specifications. The test cases are used to demonstrate that software functions are operational, input is properly accepted, output is correctly produced and the integrity of external information is maintained. It does not examine the internal logical structure of the software and finds incorrect or missing functions, interface errors, errors in data structure or external data base access, performance errors, initialization and termination errors, etc. There are many methods to efficiently carry out the black box testing [5][9].

Based on the above background, we have carried out the testing on the AOCC software in different stages.

3. AOCC SOFTWARE Testing :

"Extensive testing of the software" is taken as a major step to improve the reliability of AOCC software. In this process, we follow similar steps as mentioned in the previous section.

3.1 Unit Testing :

During unit testing, we concentrate about the correct implementation of source code, using the principle of white box testing. Each unit or procedure is tested individually ensuring that it functions properly as a unit. All paths are covered in detail. Flow, logics and computational checks are also performed. Following gives more details on each of the testing.

a) Module Interface :

The units are integrated to module, therefore checks are done to validate the I/P and O/P interface of each unit. This ensures that we encounter no problem at the module development.

b) Data Structure :

Data structures are examined to ensure that data are stored temporarily and maintain its integrity, during all the steps of program execution.

97

c) **Boundary conditions :**

Boundary conditions are checked to ensure the proper execution of unit under extreme limit conditions. All the restrictions on the units are checked and recorded.

d) **Independent path :**

The major importance at unit test is to verify the operation of all the paths. Though, it is extremely difficult to cover all possible sequences of the paths, efforts are made to the maximum extent. Beyond this stage of testing, all the error handling paths are also tested.

e) **Computation Checks :**

Another important check we are carrying out at this stage is that of computation aspects. Different precision's of arithmetic are used in practice. Also the truncational error are to be verified. This is achieved by comparing the results at various iterations in steps of say (1,10, 100, 1000,.....) for the units from the simulation to the actual realisation.

3.2 Module Testing :

The units are introduced into modules and all the paths of a module are tested as it was done in unit testing. Now the computation are done at higher level. Again like the unit test, the computation checks are performed for 1,10, 100, 1000,..... iterations. The number of computational steps are decided by the structure of the computational algorithm. The objective of this test is to continue the test until all variables of the control equation are exercised to its full dynamic range. Sometimes it may be necessary to derive special initial conditions to achieve the objective. Most of the computational errors uncovered at this stage. The execution time of the module, worst, best case and average case are measured at the end of this testing. This step is necessary to integrate the modules and maintain their time line.

3.3 Integration Testing:

This is done after integration of several modules to form a subsystem program. At this stage, all the module interfaces, execution time and the module verification are performed. The integration of module is done in bottom up approach. The timing measurement are also performed. As a thumb rule, 25% time margin expected at this level of timing evaluation.

3.4 System Testing

After the subsystem test, the software subsystem are integrated to form a full system. Several tests are conducted at this level and are discussed below.

Initially **performance tests** are performed, to verify performance of the system to meet the specification. The system is then subjected to **stress test**, where the performance of the software is observed with out-of-bound input value. The software should not misbehave in such conditions also. Then **security testing** is performed to validate the protection mechanism built into a system and validate the improper penetration in software area. Finally, **recovery testing** is performed to verify the software recovery from the fault conditions.

Unit level testing, Module & Integrated testing are carried out using our development environment. During initial stages of software development IBM RS 6000 work station with development tool which is designed in house were used as total software development platform. The development tool is specially used for the timing testing of time and module placement. In our development environment the routine gets I/P from the script file generated by the debugger. Since the debugger does not support IO operations, it is difficult to tamper the actual routine for performing IO operations.

3.5 Bench Level Test:

Here, the AOCC is interfaced with Test System as shown in fig.1. The test system simulates all the inputs that are required by AOCC. The interface circuits will be same as actual system interface in spacecraft.

The Bench Test will be carried out as per the test document. This will be a black box testing. All the functions will be validated and verified here. Each test

case will be fed and test results will be tabulated. All the problems encountered will be recorded and will go through test result review committee. Based on the review committee recommendation, further action will be taken. Once the system has undergone the bench level tests, the test engineer will certify the package performance.

3.6 Hardware-In-Loop Simulation(HILS) Test:

In HILS test, the AOCC is interfaced with actual sensor and the actuator in loop . A simulator program simulates the dynamics of the spacecraft. The simulator takes the torque output from the AOCC and provides the necessary drive signal to servo table to get the required disturbance. The sensor will be mounted on servo & will provide the errors, which is read by the AOCC and response will come in the form of torque. Fig. 2 shows all the elements of the HILS set up. Here, AOCC's dynamic behavior are verified.

3.7 Environmental Test:

The AOCC undergoes the following environmental tests after the HILS test.

1. **Thermovac test**: - The AOCC & spacecraft is subjected to various temperature under vacuum to study how it affects the various subsystems and also to monitor the spacecraft's control mechanism under these conditions.
2. **Vibration test:** - The health of various components are monitored to find in which way the vibration that occurs especially during the launch affects the AOCC & spacecraft.
3. **EMI/EMC test**:- This test will be conducted to verify the satisfactory performance of the AOCC in a radiation environment. This will be conducted only if the package is new & the design is new. EMI spikes will be introduced both in supply line as well as signal lines and system performance will be verified.

4. EXAMPLES

We present here some of the examples of various testing stages.

4.1 Earth Sensor(ES) data processing:

We take an example of ES data processing routine shown in fig 3. The purpose of this software routine is to read the ES data in AOCC and then process it for the use of controller. The unit test is performed by identifying its paths. Each of the operations are identified by number and then by using a sequence of these numbers, a flow graph is prepared as shown in fig.4 . There exist two paths named 1- 2- 3- 6- 7- 8 and 1 – 2 – 4 – 5 – 6 – 7 – 8 . The first path tests the data ready **false** condition and the second path tests the data ready **true** condition. This is an illustrative example. For more complex software, there may be many paths also.

4.2 Quarternion(Q) processing:

We present here, another example of Q computation shown in fig 5. It is a computation intensive routine. The flow graph of the routine is shown in fig.6.
It has two paths. In path 1 – 2 – 5, no computations are done. Therefore, the computational value of Q1,Q2,Q3,Q4 and q1,q2,q3,q4 are unaffected. For the path 1 – 2 – 3 – 4 – 5, the Q's and q's are computed at various iterations and the results are tabulated in a table. The computational equation & the output tabulation format are given in table – 1.

Here, the computed data are compared at different steps with simulation results.

4.3 South Sun Acquisition (SSA) Module:

We take another example of South Sun Acquisition mode, where the spacecraft is oriented such that its south face is pointed towards sun. Details of this mode may be found in [1]. Here, for the sake of understanding the testing aspect, the flow chart is given in fig. 7 and flow chart of one of the modules of SSA processing is given in fig 7.1. This subsystem is tested for performing module & integration testing. Each flow graph is a module or combination of modules. Some of the flow graphs & the paths are shown in fig. 8, 8.1. It is

presented in a top down way. At the top level, integration testing is performed at level down – module level and unit level testing are performed.

5. EXPERIENCES

Some of the problems that were detected during the various stages of testing of INSAT – 2E and corrected are listed below.

- The following were detected and corrected during the initial s/w testing :
  - During computer simulation, it was detected that there existed a matching problem due to the compiler rounding off instead of expected truncation.
  - In one of the library modules called TEST-BIT, the logical ANDing of expressions were resulting in incorrect outputs
  - Due to improper assembler directive, some programs were incorrectly allocated to RAM instead of PROM
  - Due to some mismatch of the requirement specification , the relative Q computation was done every 151th cycle instead of every 150 cycles. This was found during simulation
- The following were detected and corrected during the bench level testing:
  - During initial bench level test, it was found that there was floating point truncation error
  - There was use of incorrect variables in the tacho hysterisis. This was detected during code walk-through
  - The flap timing requirement was not met & the placement of module was changed
  - The polarity check for SADA(Solar Array Drive Assembly) direction bit reversal was not correct due to improper specification
  - Gain pointer update was done wrongly in Telecommand processing
  - It was found that floating point to integer conversion resulted in overflow

- Due to inhibit duration, incorrect updates were observed for the wheel speed

6. CONCLUSION

This paper presents testing aspect of the AOCC software. The testing is performed in several steps. Such a kind of exhaustive testing is necessary for th critical application. Most of the computational and logical errors are uncovered by this testing. Each stage of testing needs a thorough analysis of software and proper sequence of test case generation is expected. INSAT – 2 E AOCC software testing is performed in this manner. No major errors have been found either at ground or at on-board.

6. REFERENCES:

1. "INSAT-2E AOCE Software Design Document", Internal report, Control Systems Group, ISRO Satellite Centre (ISAC), Bangalore.
2. "AOCS interface control document", Internal report, Control Systems Group, ISRO Satellite Centre (ISAC), Bangalore.
3. S. Satheesh, S.Manjunatha Prasanna, V.K. Agrawal, "Software Structure for IRS-1C Attitude and Orbit Control System", National Workshop on flight Software, Trivandrum, June 1996, pp 1.6 – 1.16.
4. V.K.Agrawal, M.M. Nayak, K.M. Bharadwaj, "Software Development for Attitude and Orbit Control Computer – A Methodology", National Workshop on flight Software, Trivandrum, June 1996, pp 2.28 – 2.43.
5. Roger S. Pressman, "Software Engineering – A Practitioner's Approach, Fourth Edition, McGraw Hill International Edition, 1997.
6. S. Balaji, S.Manjunatha Prasanna, C. Rathna, V.K. Agrawal, Control Electronics Division, ISRO Satellite Centre, Bangalore-560 017 "Software Development for Spacecraft Onboard System Using a High level Language" - Paper published in the Journal of Spacecraft Technology, Vol. 6, No. 1, January 1996, pp 17 – 24.

7. "INSAT – 2E configuration document", Internal report, ISRO Satellite Centre (ISAC), Bangalore.
8. "Test & Evaluation Report of AOCE", Internal report, ISRO Satellite Centre (ISAC), Bangalore.
9. "Bus Management Unit for SMALLSAT", Internal report, Control Systems Group, ISRO Satellite Centre (ISAC), Bangalore.

Acknowledgement

The authors gratefully acknowledge the guidance provided by Dr. P.S Goel, Director, ISRO Satellite Centre, Bangalore during the course of this work.

TABLE – 1

Computational Equations for Q Processing:

q_1 = Constant * Incremental Yaw Angle

q_2 = Constant * Incremental Roll Angle

q_3 = Constant * Incremental Pitch Angle

$q_4 = 1.0 - 3.33E-01 * (q_1^2 + q_2^2 + q_3^2)$

$$\begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix} = \begin{bmatrix} q_4 & q_3 & -q_2 & q_1 \\ -q_3 & q_4 & q_1 & q_2 \\ q_2 & -q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_{31} & q_4 \end{bmatrix} * \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \end{bmatrix}$$

Output Tabulation Format:

Step No	Simulated q's				Observed q's				Simulated Q's				Observed Q's			
	q1	q2	q3	q4	q1	q2	q3	q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4

105

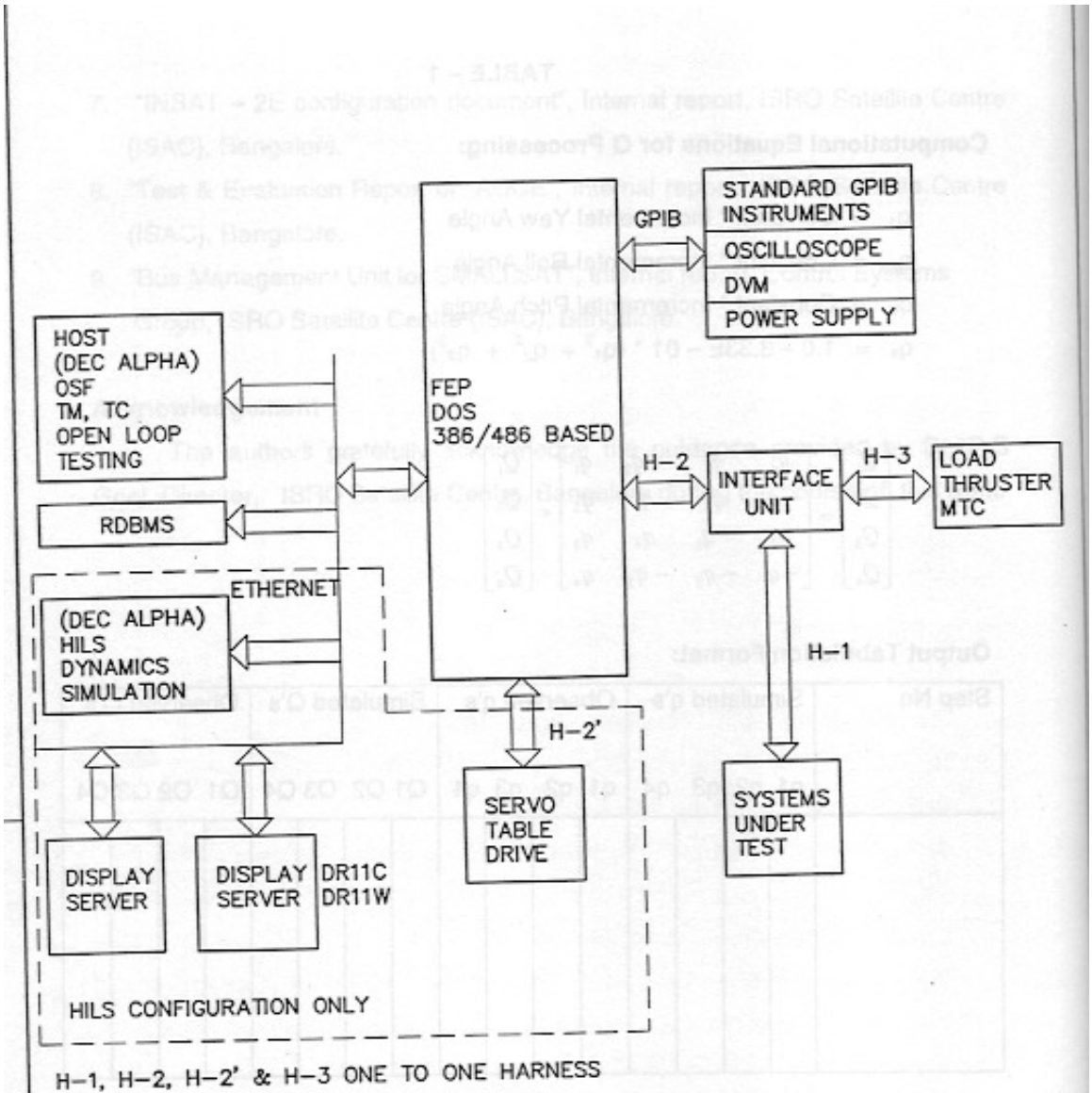


FIG.1 TEST SYSTEM SETUP

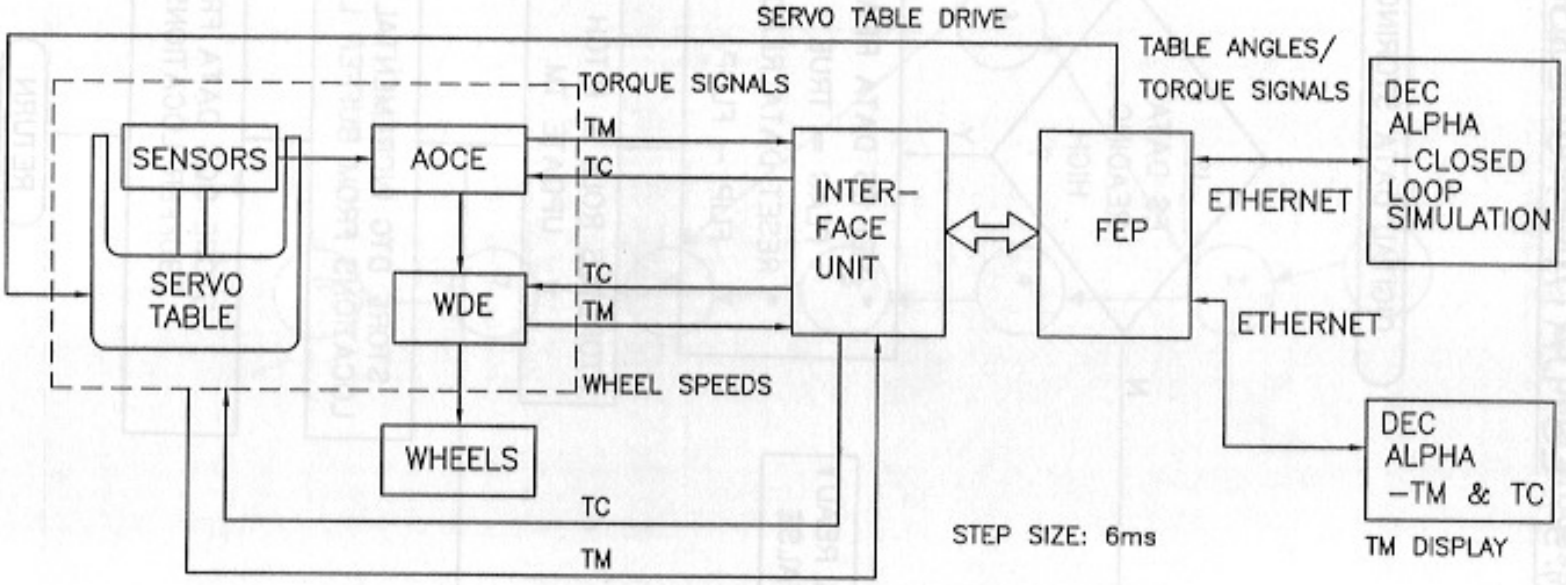


FIG. 2 HILS SET–UP

Fig. 3 ES DATA PROCESSING

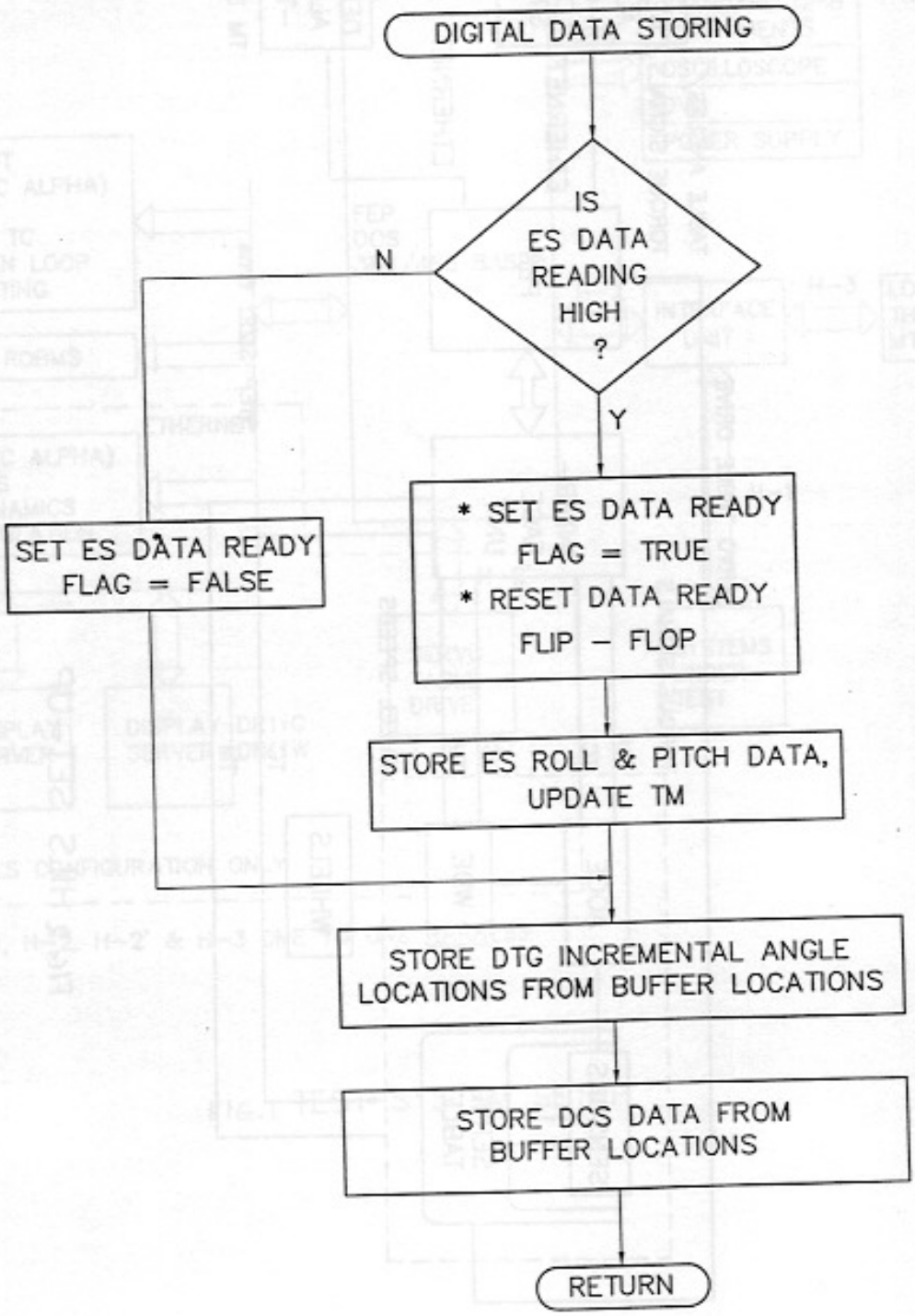


Fig.4 **Flow graph notation of ES data processing**

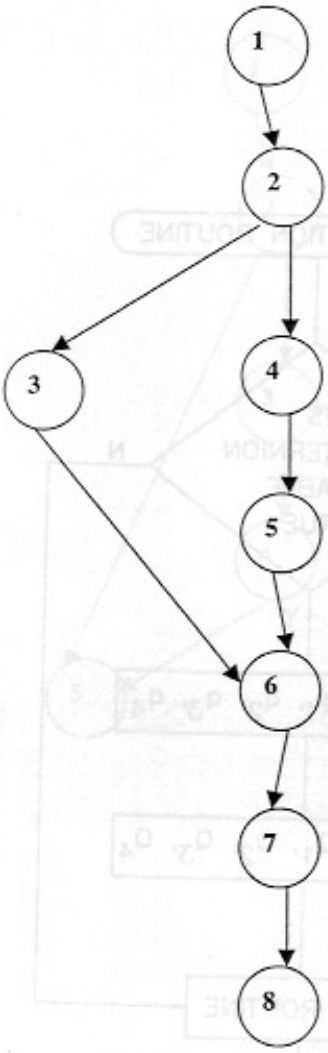
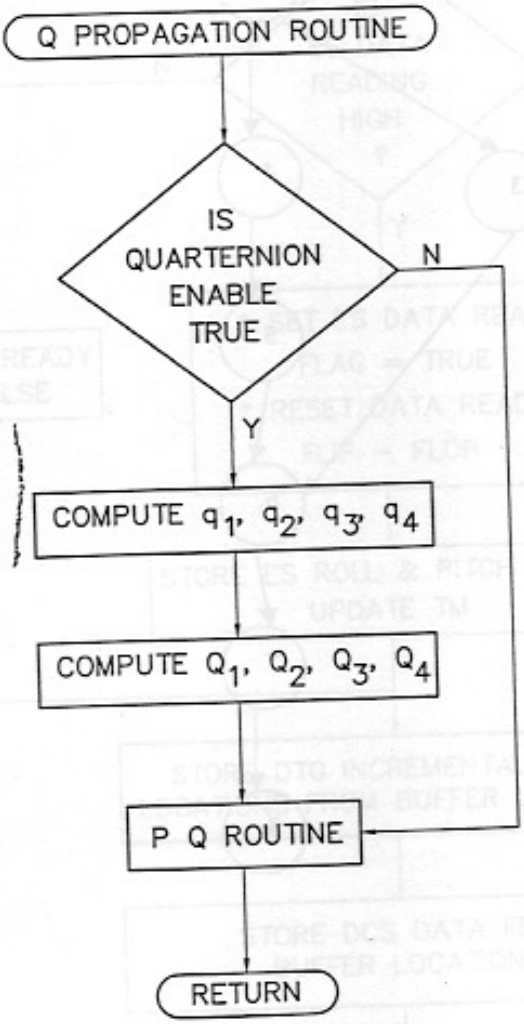


Fig.5 QUARTERNION PROPAGATION



110

Fig.6 **Flow graph notation of Quarternion propagation**

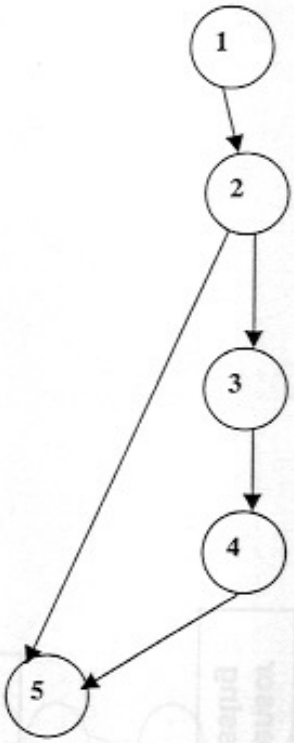
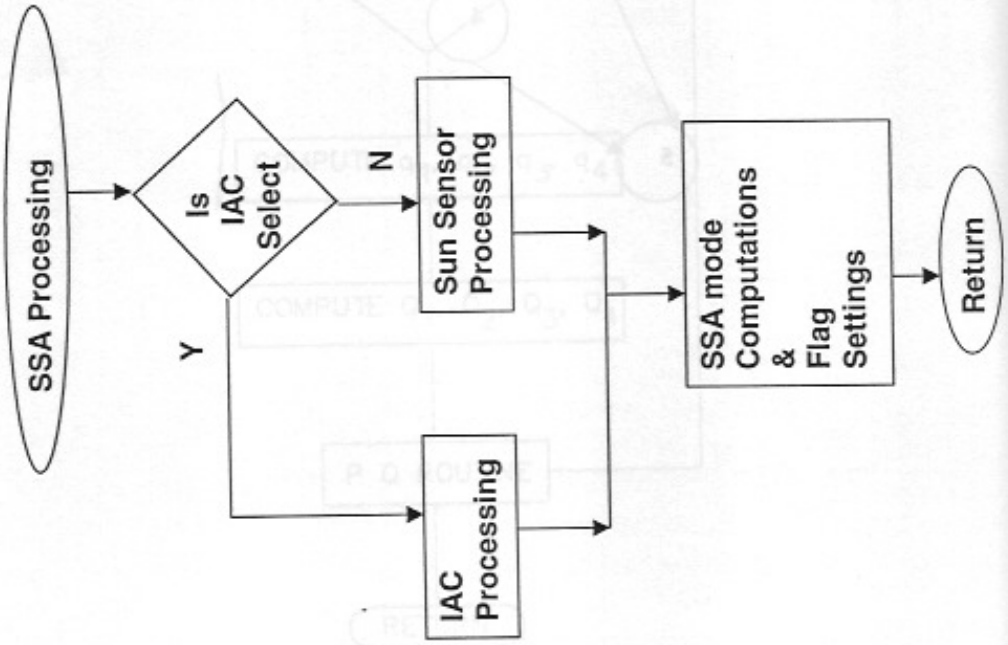
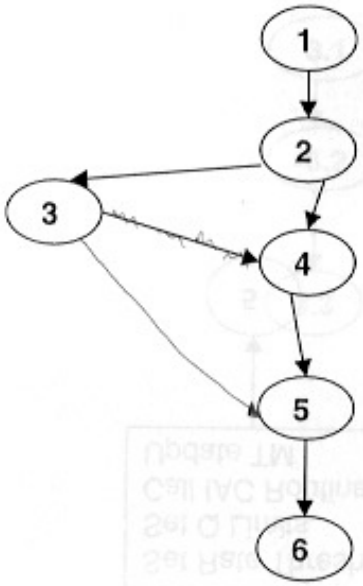


Fig. 7 SOUTH SUN ACQUISITION PROCESSING



112

Fig. 8 Flow Graph Notation of SSA Processing



Paths:

Path1 : 1-2-3-5-6

Path2 : 1-2-4-5-6

Cyclomatic complexity:

V(G) = E-N+2 = P+1

= 6-6+2 = 1+1

= 2

Fig. 7.1 IAC Processing

Fig. 8.1 Flow Graph Notation of fig7.1

Paths:

Path1 : 3.1-3.2-3.3

Cyclomatic complexity:

V(G) = E-N+2 = P+1
= 2-3+2 = 0+1
= 1